CS380: Introduction to Computer Graphics

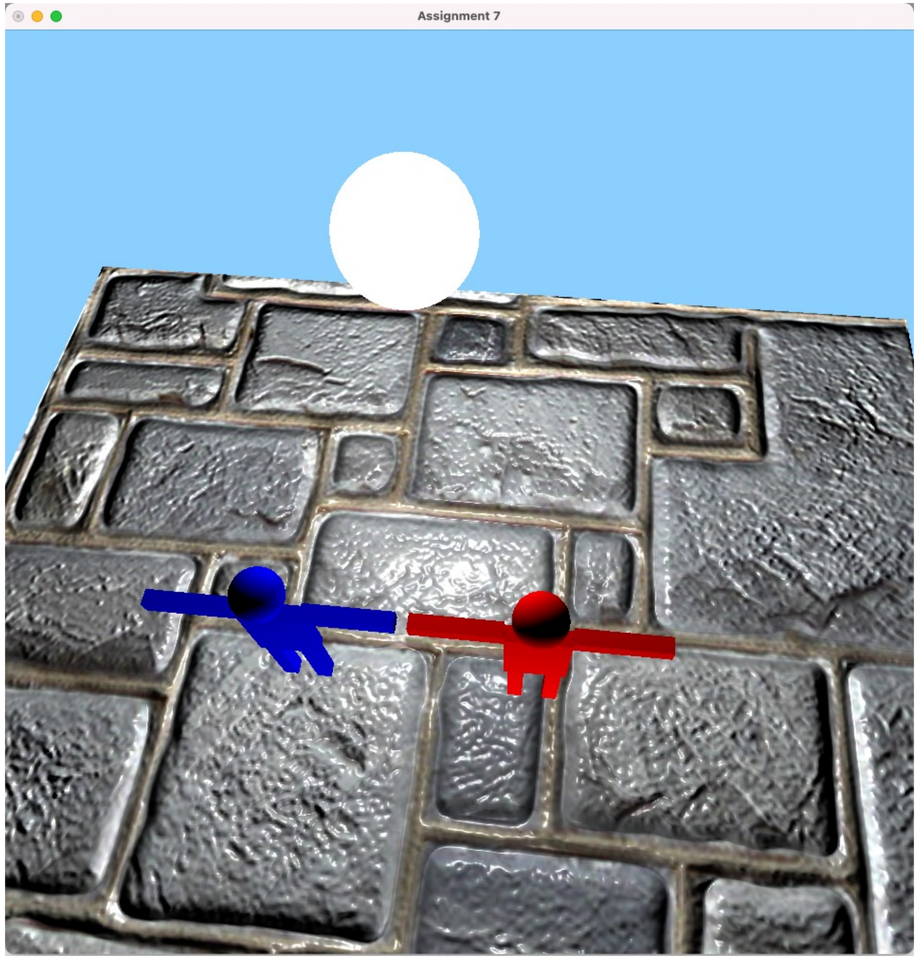
Material System

LAB SESSION 7
KUNHO KIM

Spring 2023
KAIST

Overview

- Material infrastructure.
  - Multiple shaders per one frame.
- Bump mapping.
  - Normal map.
- **Bonus (not mandatory).**
  - Shadow mapping.



Goals

- **Task 1**: Carefully read the PDF file to understand the material infrastructure.
  - Then, proceed with migrating the codebase.

- **Task 2**: Implement bump mapping.
  - Make two lights pickable and movable.
  - Modify the shader file for the normal map.

Preparation

- Assignment 7 will be built upon your **assignment 4** codebase.
- Download the assignment 7 code files and integrate them with your **assignment 4** codebase.
- In case there are files with duplicate names, replace them with the new files.

Preparation

These are the files that **need to be replaced**:

- “asstcommon.h”
- “drawer.h”
- “picker.cpp”
- “picker.h”
- “scenegraph.cpp”
- “scenegraph.h”

Preparation

These are the new files:

- “asst7-snippets.cpp”
- “uniforms.h”
- “geometry.cpp”
- “geometry.h”
- “material.cpp”
- “material.h”
- “renderstates.cpp”
- “renderstates.h”
- “Fieldstone.ppm”
- “FieldstoneNormal.ppm”
- “shaders/normal-gl2.vshader”
- “shaders/normal-gl2.fshader”
- “shaders/normal-gl3.vshader”
- “shaders/normal-gl3.fshader”

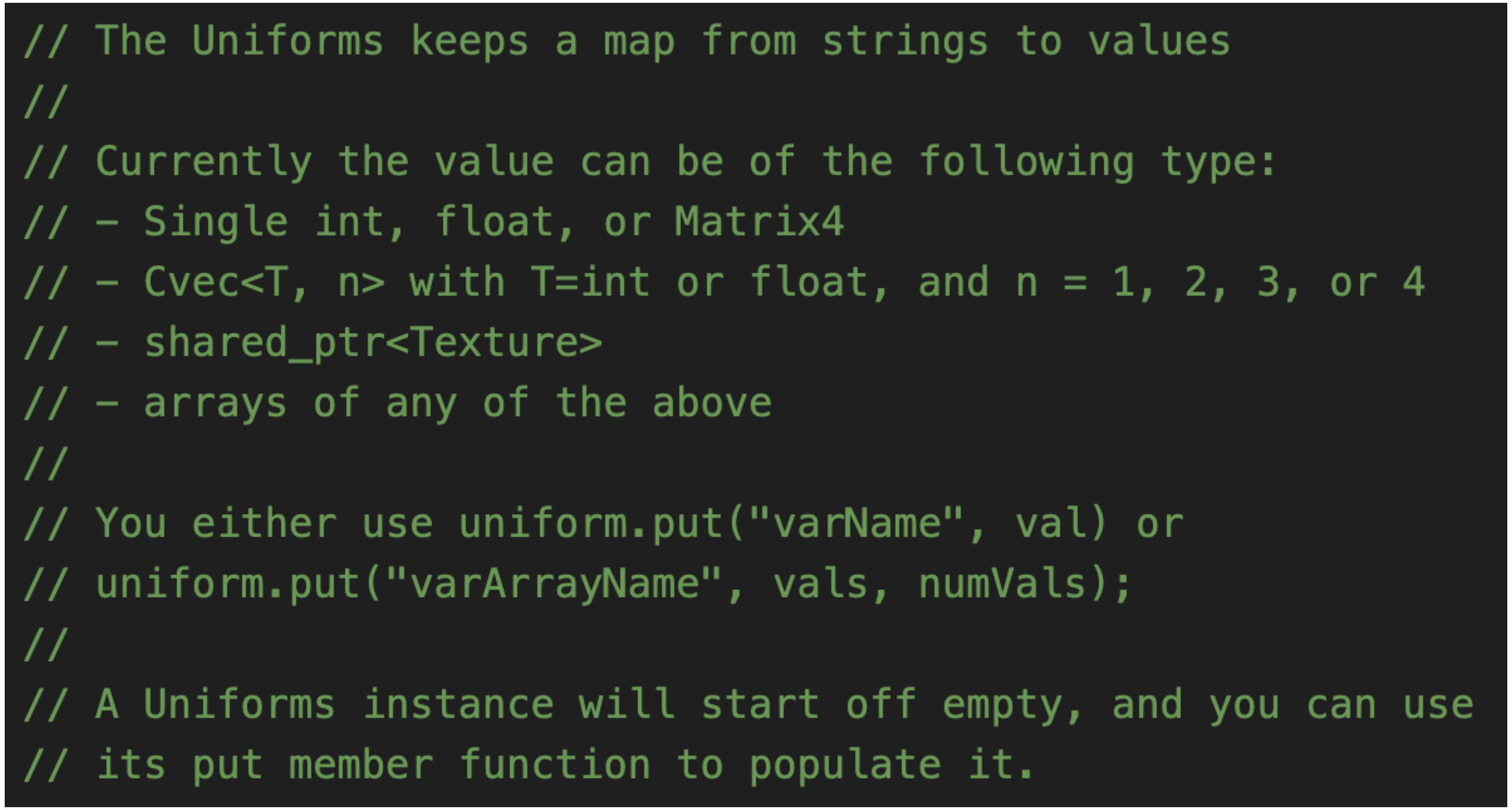
Task 1 – Material Infrastructure

Multiple Shaders

- Each shader has its own uniform variables.
- Different GLSL shaders do not know the values of the other shader's uniform variables.
- This suggests we need some kind of data structure to hold the values of these uniform variables.

Transfer Uniform Value

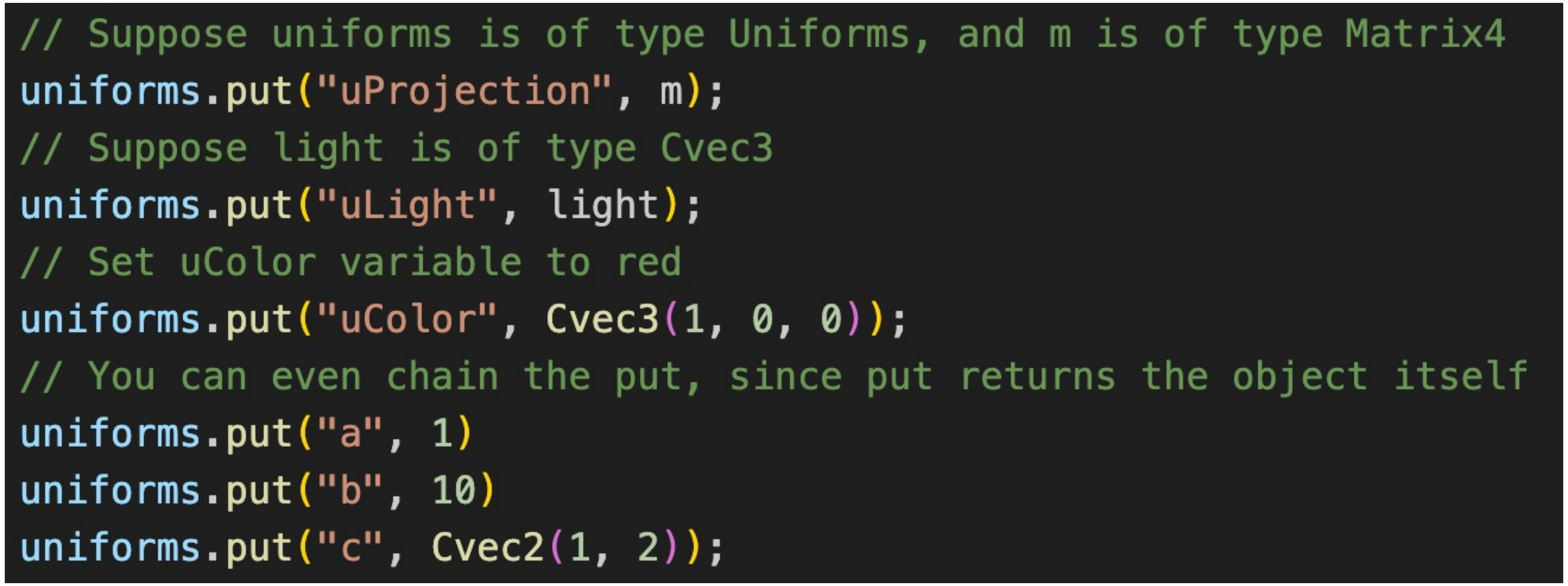
`Uniforms` is a dictionary mapping from names to values.



```
// The Uniforms keeps a map from strings to values
//
// Currently the value can be of the following type:
// - Single int, float, or Matrix4
// - Cvec<T, n> with T=int or float, and n = 1, 2, 3, or 4
// - shared_ptr<Texture>
// - arrays of any of the above
//
// You either use uniform.put("varName", val) or
// uniform.put("varArrayName", vals, numVals);
//
// A Uniforms instance will start off empty, and you can use
// its put member function to populate it.
```

Transfer Uniform Value

`Uniforms` class is defined in “`uniforms.h`”.



```
// Suppose uniforms is of type Uniforms, and m is of type Matrix4
uniforms.put("uProjection", m);
// Suppose light is of type Cvec3
uniforms.put("uLight", light);
// Set uColor variable to red
uniforms.put("uColor", Cvec3(1, 0, 0));
// You can even chain the put, since put returns the object itself
uniforms.put("a", 1)
uniforms.put("b", 10)
uniforms.put("c", Cvec2(1, 2));
```

Remove ShaderState

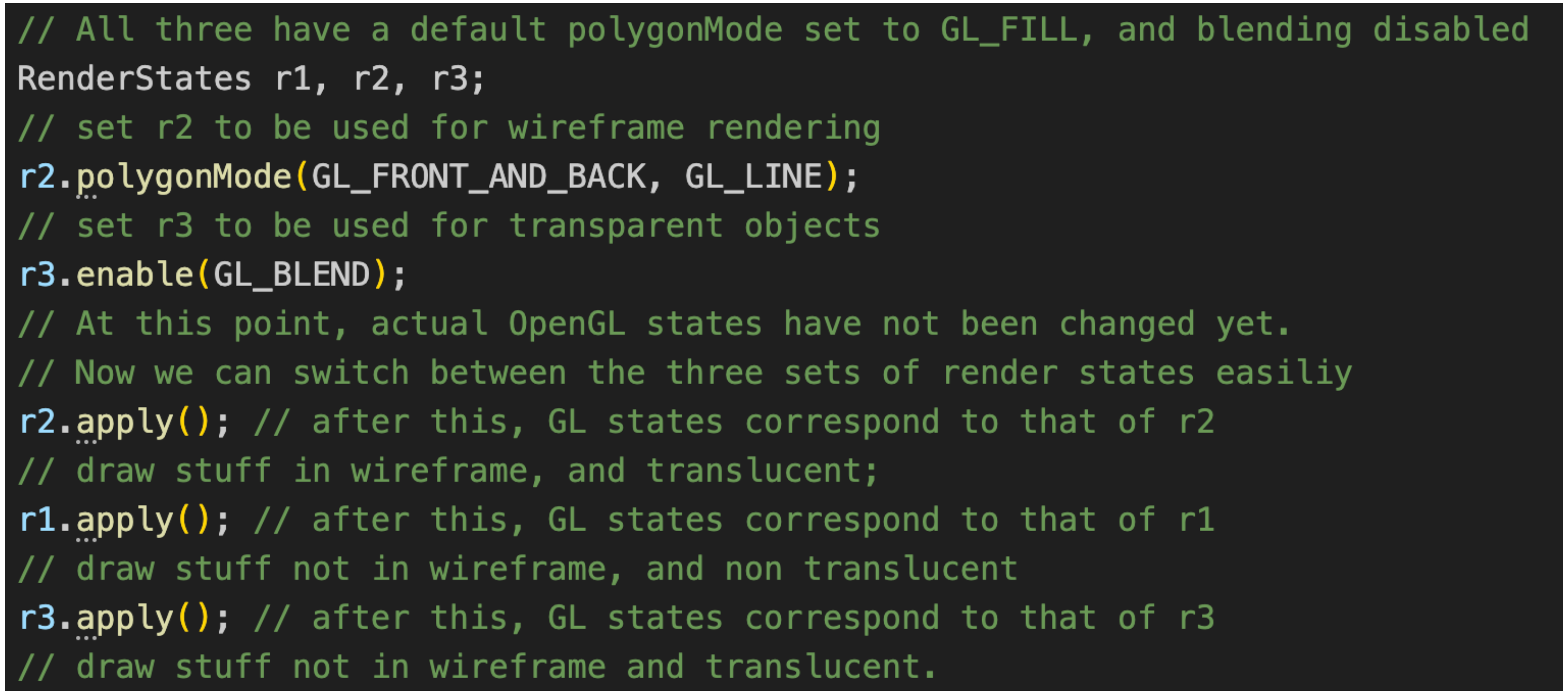
- So far we have been using only one GLSL shader per frame during our rendering, as controlled by the global variable `g_activeShader`, an index into the global array of `ShaderState`'s `g_shaderStates`[].
- From now on, we will use `Uniforms` instead of `ShaderState`.

RenderStates

- `RenderStates` class is defined in “`renderstates.h`”.
- `RenderStates`: A subset of OpenGL state.
- The state does not immediately take effect in OpenGL.
- The state will be applied when you call the member function: `apply()`.

RenderStates

`RenderStates` is useful to manage multiple shaders.

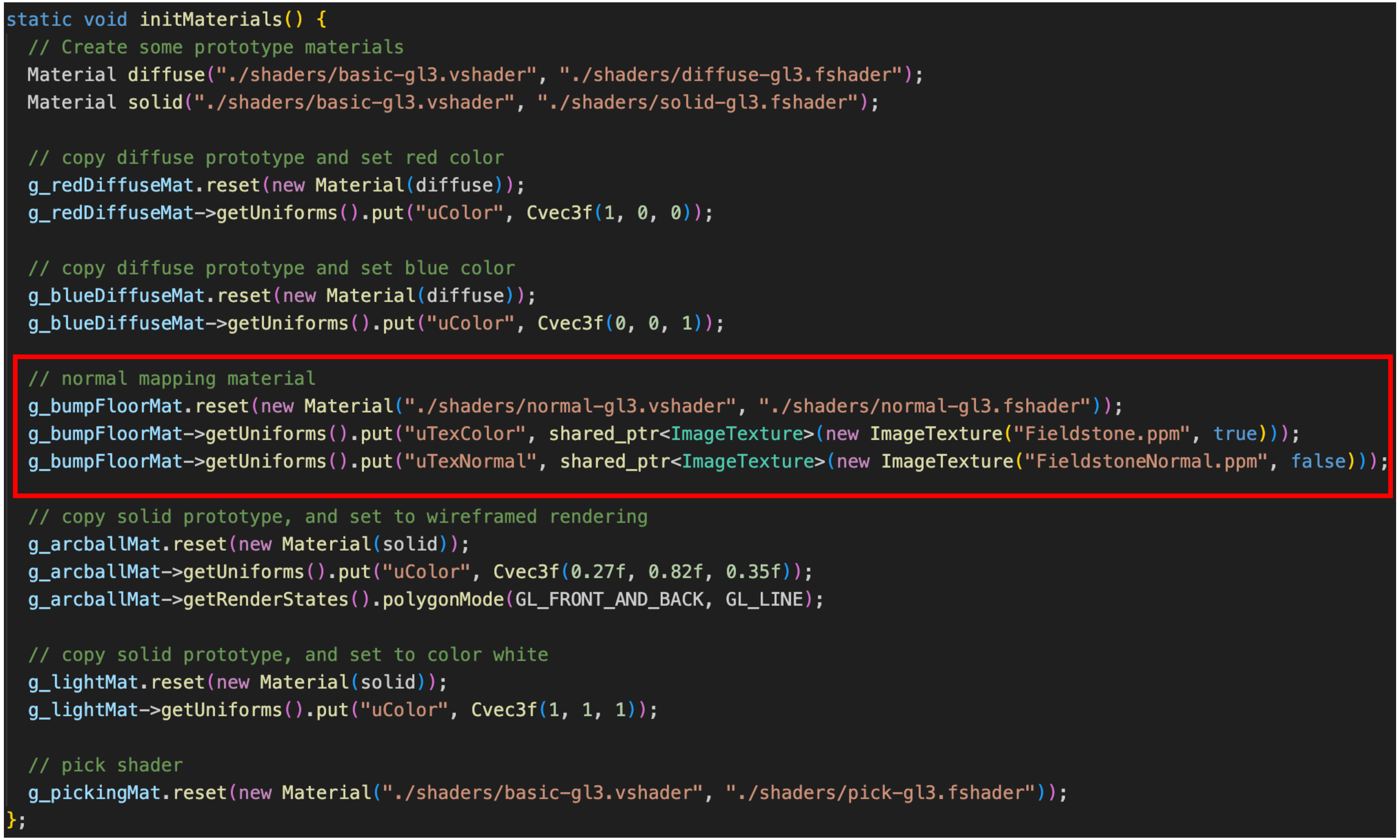


```
// All three have a default polygonMode set to GL_FILL, and blending disabled
RenderStates r1, r2, r3;
// set r2 to be used for wireframe rendering
r2.polygonMode(GL_FRONT_AND_BACK, GL_LINE);
// set r3 to be used for transparent objects
r3.enable(GL_BLEND);
// At this point, actual OpenGL states have not been changed yet.
// Now we can switch between the three sets of render states easiliy
r2.apply(); // after this, GL states correspond to that of r2
// draw stuff in wireframe, and translucent;
r1.apply(); // after this, GL states correspond to that of r1
// draw stuff not in wireframe, and non translucent
r3.apply(); // after this, GL states correspond to that of r3
// draw stuff not in wireframe and translucent.
```

Geometry and Texture

- Geometry (in "`geometry.h`"):
  - `GeometryPN`: position and normal.
  - `GeometryPNTBX`: position, normal, tangent, binormal, and texture coordinate.
- Texture (in "`texture.h`"):
  - `ImageTexture`: A `Texture` that loads a PPM file with three channels and optionally stores the content in sRGB color space.

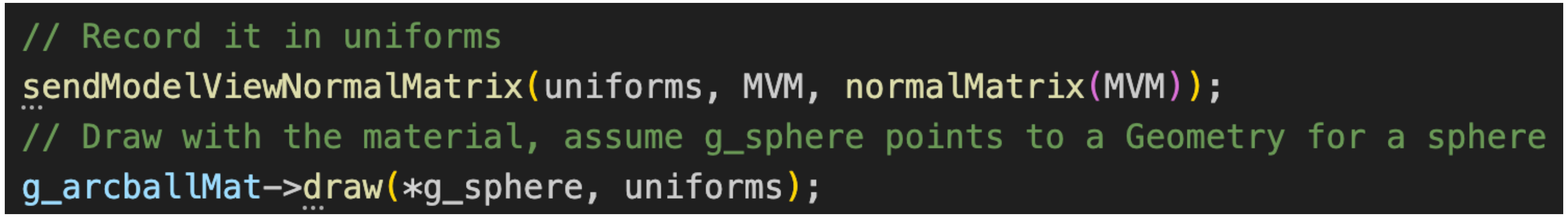
Load Texture



```
static void initMaterials() {
  // Create some prototype materials
  Material diffuse("./shaders/basic-gl3.vshader", "./shaders/diffuse-gl3.fshader");
  Material solid("./shaders/basic-gl3.vshader", "./shaders/solid-gl3.fshader");

  // copy diffuse prototype and set red color
  g_redDiffuseMat.reset(new Material(diffuse));
  g_redDiffuseMat->getUniforms().put("uColor", Cvec3f(1, 0, 0));

  // copy diffuse prototype and set blue color
  g_blueDiffuseMat.reset(new Material(diffuse));
  g_blueDiffuseMat->getUniforms().put("uColor", Cvec3f(0, 0, 1));

  // normal mapping material
  g_bumpFloorMat.reset(new Material("./shaders/normal-gl3.vshader", "./shaders/normal-gl3.fshader"));
  g_bumpFloorMat->getUniforms().put("uTexColor", shared_ptr<ImageTexture>(new ImageTexture("Fieldstone.ppm", true)));
  g_bumpFloorMat->getUniforms().put("uTexNormal", shared_ptr<ImageTexture>(new ImageTexture("FieldstoneNormal.ppm", false)));

  // copy solid prototype, and set to wireframed rendering
  g_arcballMat.reset(new Material(solid));
  g_arcballMat->getUniforms().put("uColor", Cvec3f(0.27f, 0.82f, 0.35f));
  g_arcballMat->getRenderStates().polygonMode(GL_FRONT_AND_BACK, GL_LINE);

  // copy solid prototype, and set to color white
  g_lightMat.reset(new Material(solid));
  g_lightMat->getUniforms().put("uColor", Cvec3f(1, 1, 1));

  // pick shader
  g_pickingMat.reset(new Material("./shaders/basic-gl3.vshader", "./shaders/pick-gl3.fshader"));
};
```

Material

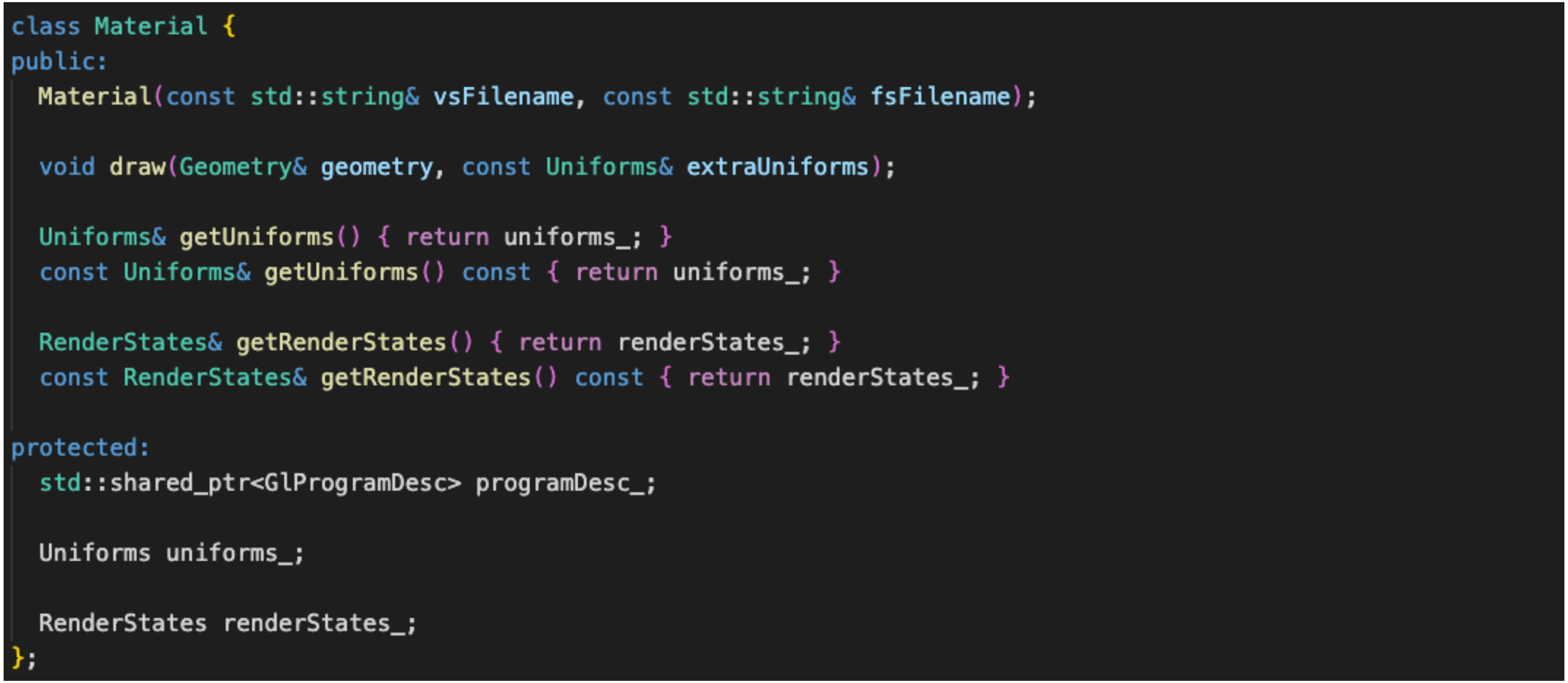
- The actual `Material` class contains three parts.
  - `shared_pointer`: GLSL shader program used.
  - `Uniforms`: accessible through `getUniforms`().
  - `RenderStates`: accessible through `getRenderStates`() .
- Member function:
  - `draw(geometry, extraUniforms).`



```
// Record it in uniforms
sendModelViewNormalMatrix(uniforms, MVM, normalMatrix(MVM));
// Draw with the material, assume g_sphere points to a Geometry for a sphere
g_arcballMat->draw(*g_sphere, uniforms);
```

Material

`Material` class is defined in “`material.h`”.



```
class Material {
public:
  Material(const std::string& vsFilename, const std::string& fsFilename);

  void draw(Geometry& geometry, const Uniforms& extraUniforms);

  Uniforms& getUniforms() { return uniforms_; }
  const Uniforms& getUniforms() const { return uniforms_; }

  RenderStates& getRenderStates() { return renderStates_; }
  const RenderStates& getRenderStates() const { return renderStates_; }

protected:
  std::shared_ptr<GlProgramDesc> programDesc_;

  Uniforms uniforms_;

  RenderStates renderStates_;
};
```

Scene Graph & Material

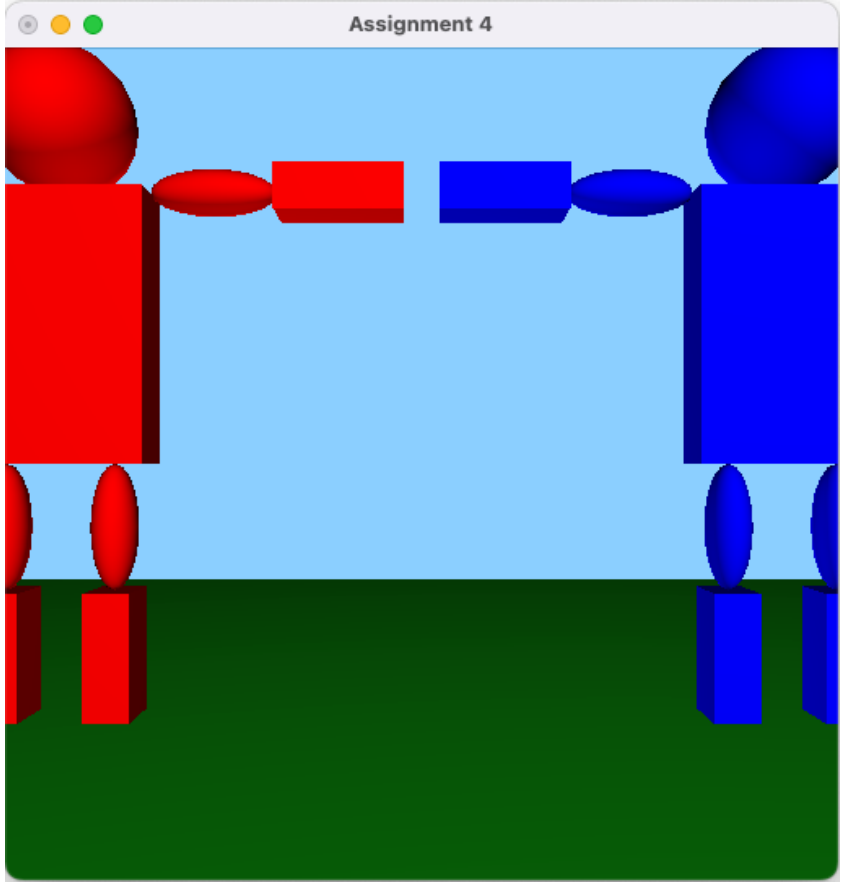
Each `SgGeometryShapeNode` has own `Material`:

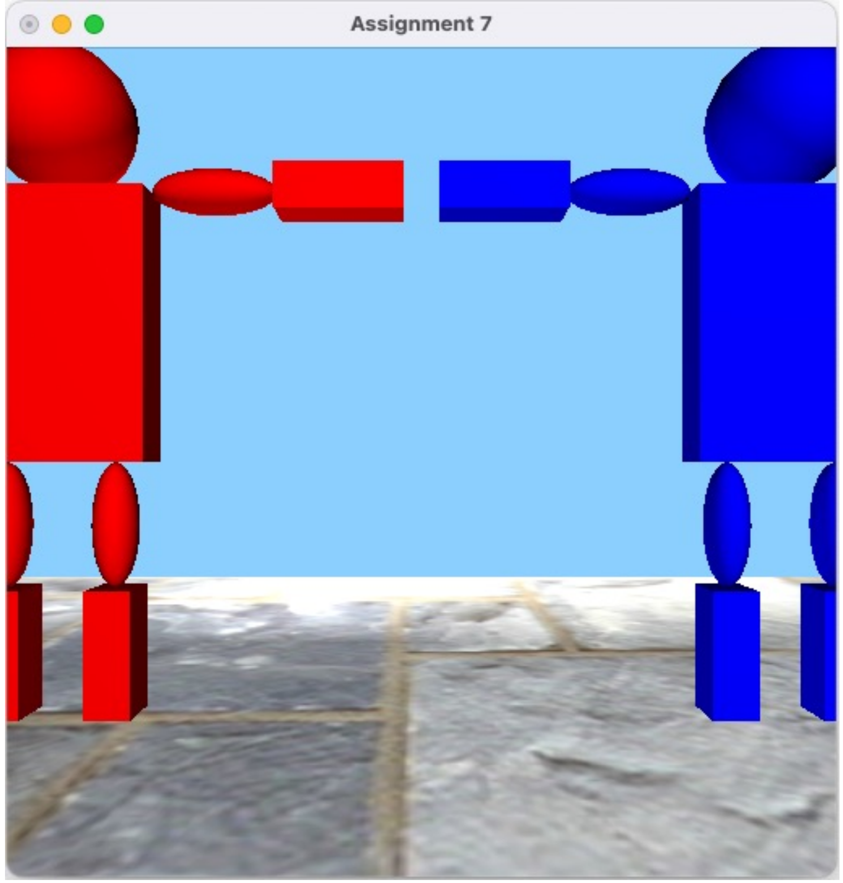
- The robots: diffuse color.
- The arcball: wireframe and solid color.
- The ground: texture color and normal.
- The lights: solid color.

Code Migration

Follow the instructions in “`asst7-snippets.cpp`” to change your code to use the new `Material` system.

Result After Task 1





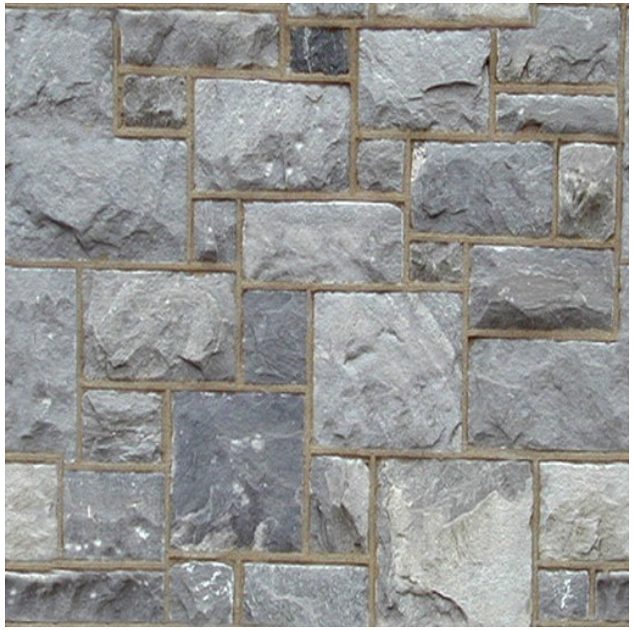
Task 2 – Bump Mapping

Make Two Lights

- The first goal is to make the two lights of the scene pickable and movable.
- Make two sphere lights with reference to the `initScene()`, similar to what you made in the previous assignments.
- Light nodes must be children of root node, `g_world`.

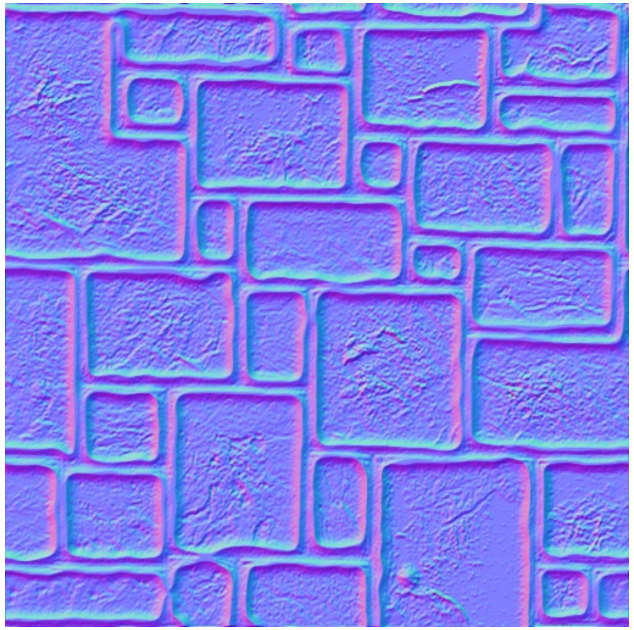
What is Bump Mapping?

Instead of changing the geometry itself, modify the surface normal to simulate bumps.



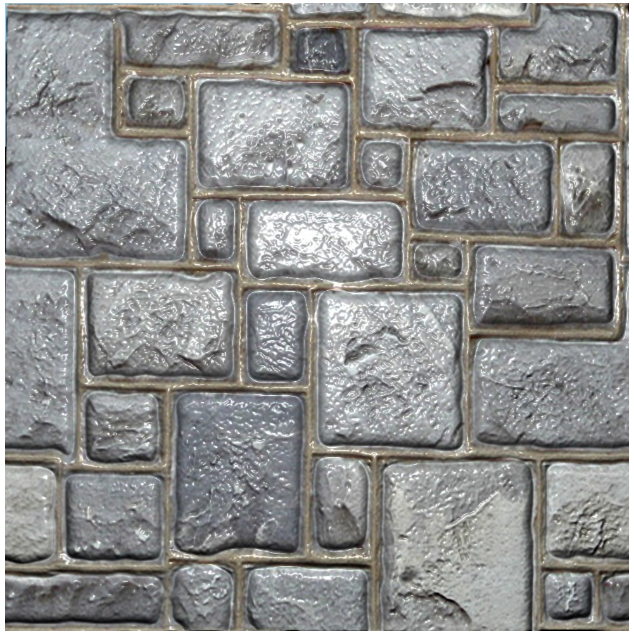
Simple texture

+



Normal Map

=

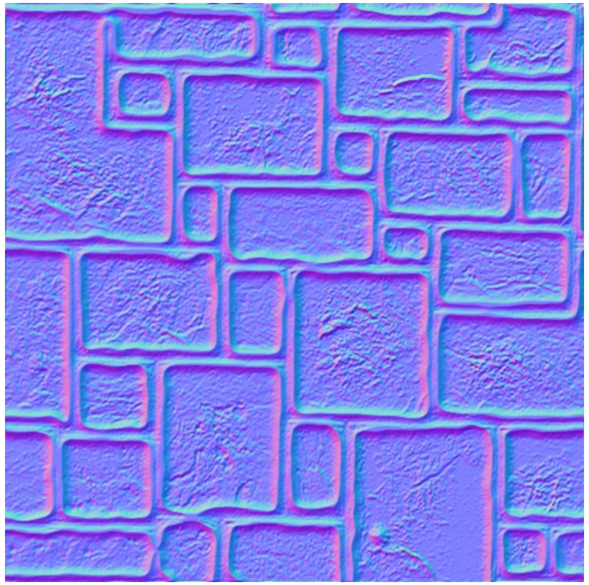


Bump

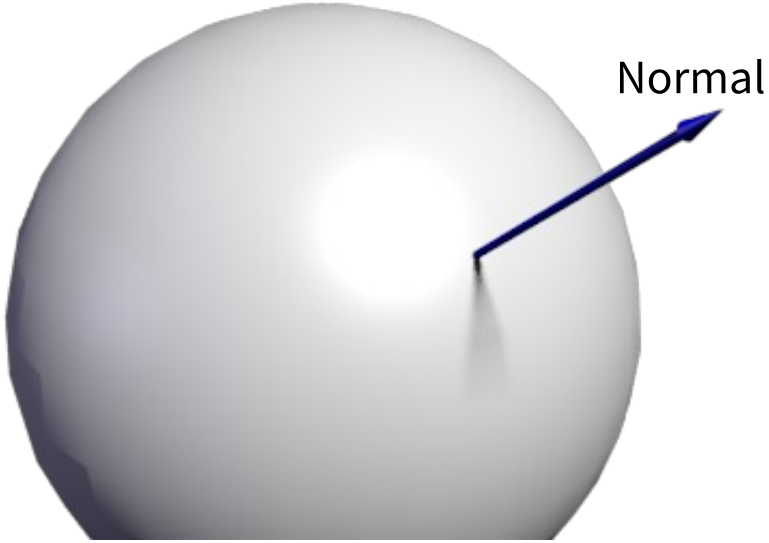
Why Do We Need Bump Mapping?

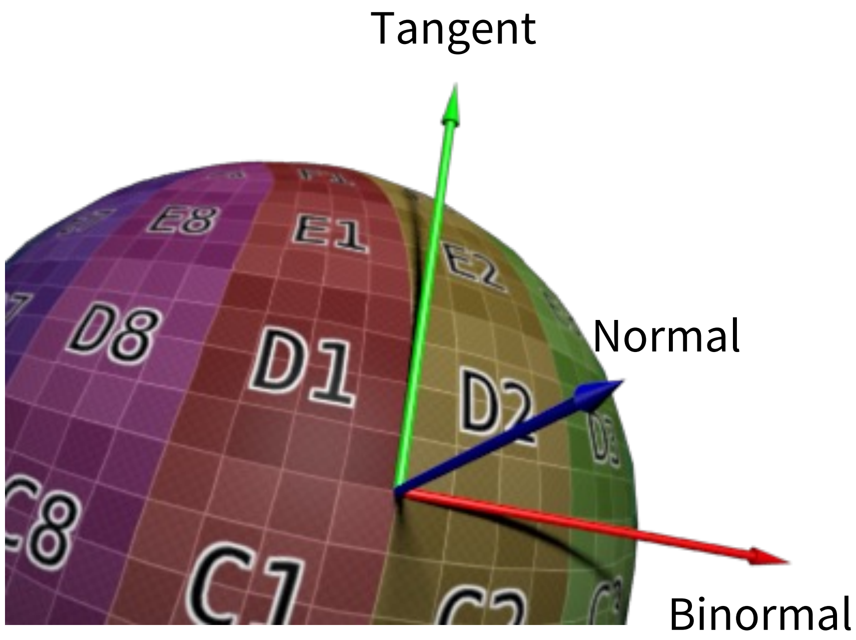
- The reason for using the normal map is that it takes a huge number of polygons to express the bumps of an object in polygons.
- However, it is so heavy that it needs to use a limited number of polygons but describe more plausible objects.
- Instead of changing the geometry itself, we can modify the surface normal to simulate bumps.

Tangent Frame



Normal Map





http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/

Bump Mapping

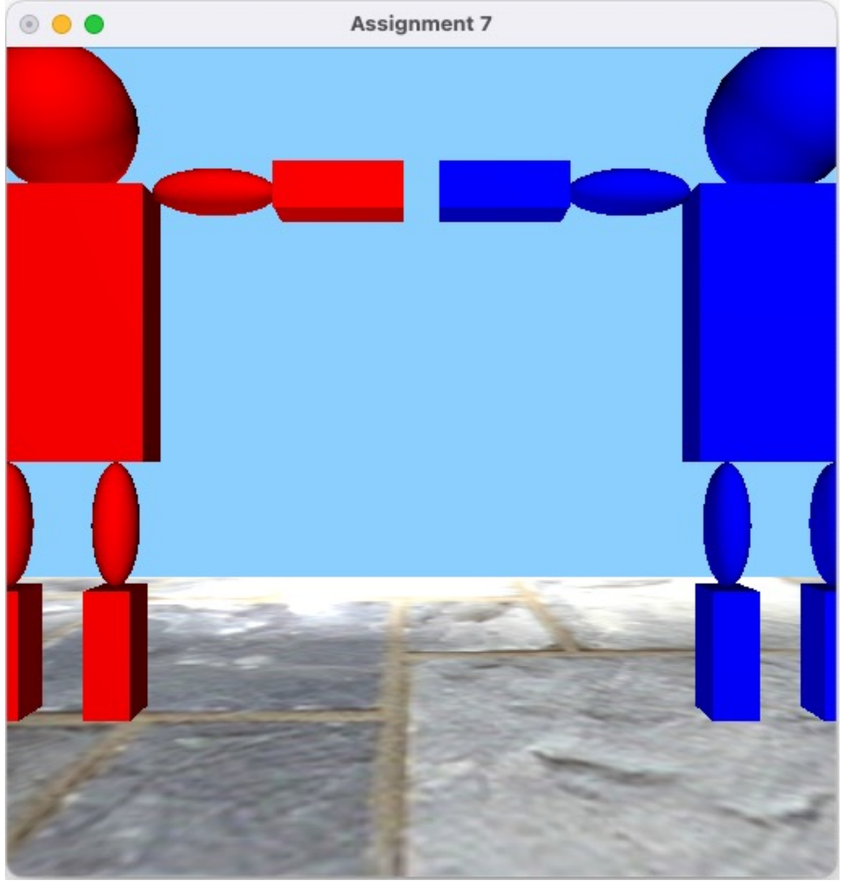
Normal map is defined w.r.t. the tangent frame.

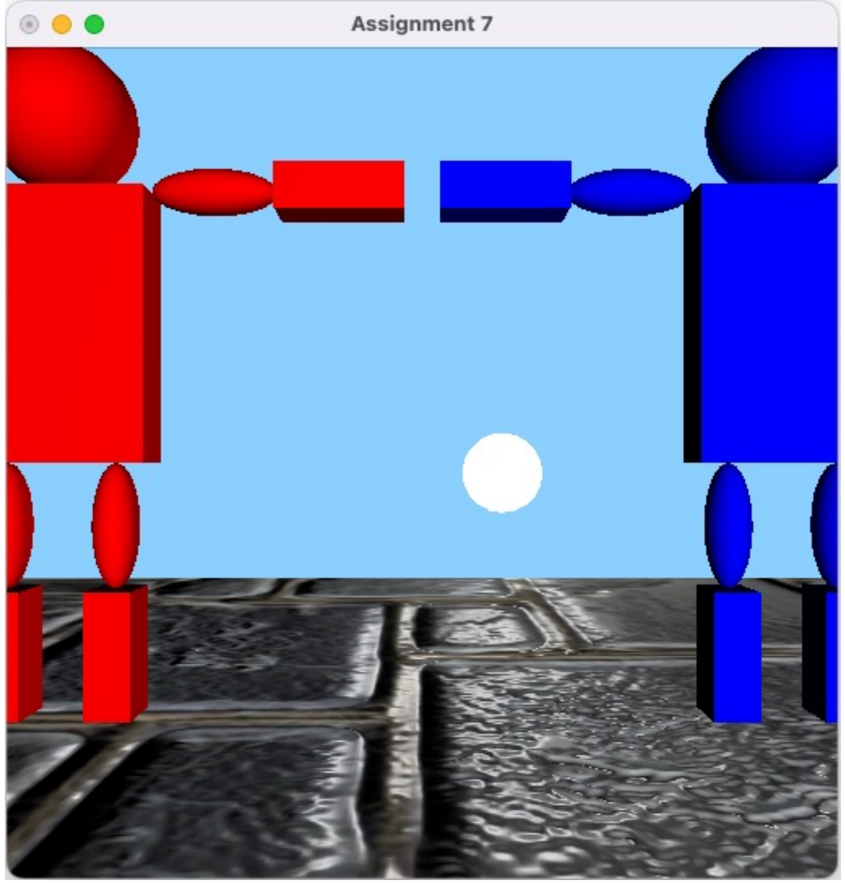
- $T = \ [\text{tangent, binormal, normal}]$.
- Object frame: $\vec{\mathbf{b}}^{\boldsymbol{t}} = \vec{\mathbf{e}}^{\boldsymbol{t}} M$.
- Tangent frame: $\vec{\mathbf{t}}^{\boldsymbol{t}} = \vec{\mathbf{b}}^{\boldsymbol{t}} T$.
- Normal in tangent frame: $\boldsymbol{n} = \left[n_r, n_g, n_b, 0\right]^t$.
- Shading computation: $\text{dot}(\text{normalize}(M^{-t} T \boldsymbol{n}),\ \text{normalize}(\boldsymbol{v}))$.

GLSL

- Texture coordinates: each vertex will need (x, y) texture coordinates. You can access them as `vTexCoord` in the fragment shader.
- The texture stores its data as real numbers between [0, 1], while normal coordinates are in [-1, 1]. Thus we need to apply a scale and then shift to the data before using it.
- We pass $M^{-t}T$ as a varying variable called `vNTMat` to the fragment shader.

Result After Task 2





ground texture changed

TODOs

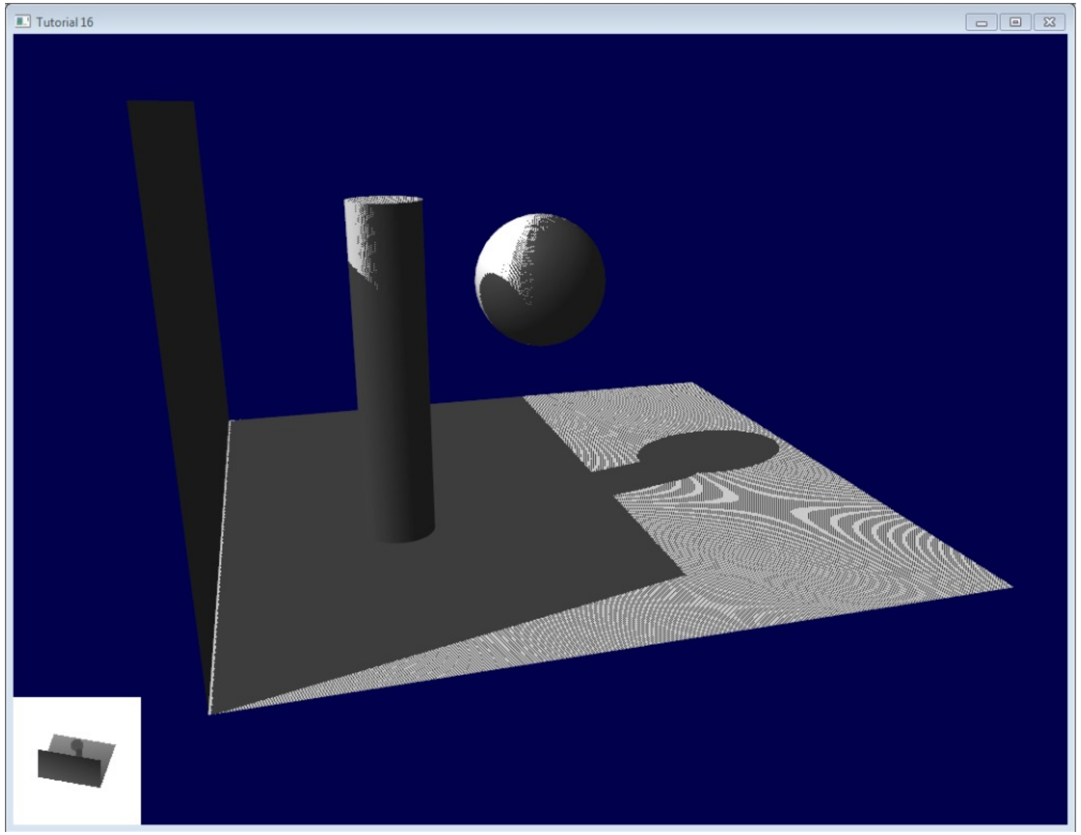
- **Task 1:** Read the PDF file carefully and understand the material infrastructure.
  - Then, migrate the code base.

- **Task 2:** Implement bump mapping.
  - Make pickable and movable two lights.
  - Modify the shader file for the normal map.

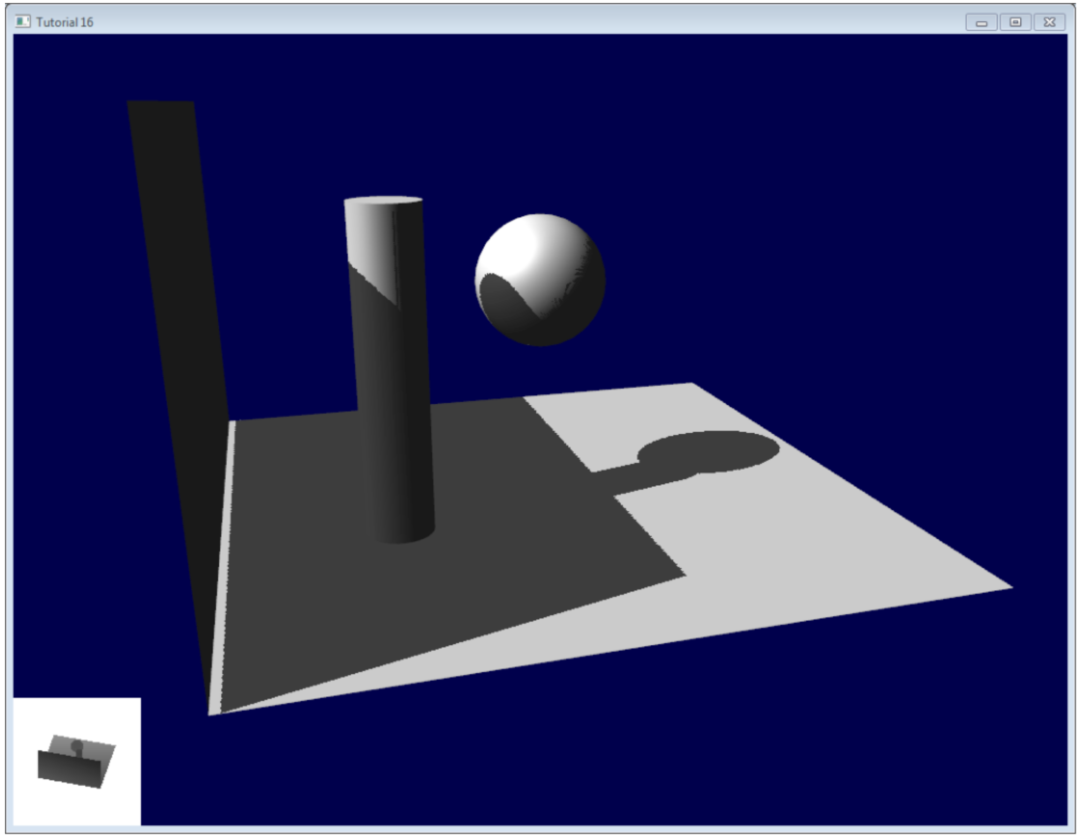
Evaluation

- Has the ground texture changed?
- Are there two lights on the scene? (above the ground)
- Are the two lights pickable and movable?
- Is bump mapping implemented correctly?

Bonus Assignment: Shadow Mapping



Moiré



http://www.opengl-tutorial.org/kr/intermediate-tutorials/tutorial-16-shadow-mapping/

Bonus Assignment: Shadow Mapping

- The bonus assignment is 50 points (half of one assignment).

- This makes up for the points you lost in the other assignments (but NOT in the exams and participation scores).

Bonus Assignment: Shadow Mapping

Only for the bonus assignment, you are allowed to use any reference and AI assistant tools. You can borrow code from GitHub repos and use Copilot and ChatGPT.

However, it must be implemented based on your solution of assignment 7.

Bonus Assignment: Evaluation

- Is the correct shadow created on the robot when the lighting is positioned differently?

- Is there no moiré?

Submission

solution binary를 올려주진 않음 (shader 건들여야해서 숨길수가 없음)

bonus assignment 하면 좋겠음, 좋은 경험이 될거 같음

- internet 코드 봐도 됨, copilot 등
- asst7 위에 작업해야함 적용시켜서...

- Due: Sun, Jun 4 23:59 KST.
- Late submission: Up to two days (~Tues, Jun 6 23:59 KST) with a penalty of 20% of the score.
- Compress the codes in a .zip file and submit it on Gradescope.
- **If you solved the bonus assignment, add README.md to your .zip file and explain your code for the shadow mapping.**